SoCQ: a Pervasive Environment Management System

Yann Gripay
Université de Lyon, INSA-Lyon,
LIRIS – UMR 5205 CNRS
F-69621 Villeurbanne, France
yann.gripay@liris.cnrs.fr

Frédérique Laforest
Université de Lyon, INSA-Lyon,
LIRIS – UMR 5205 CNRS
F-69621 Villeurbanne, France
frederique.laforest@liris.cnrs.fr

Jean-Marc Petit
Université de Lyon, INSA-Lyon,
LIRIS – UMR 5205 CNRS
F-69621 Villeurbanne, France
jean-marc.petit@liris.cnrs.fr

RESUME
Les systèmes pervasifs donnent un aperçu de ce que seront les environnements digitaux du futur. SoCQ prend une perspective orientée données de l'environnement pervasif par l'intermédiaire d'une vue unifiée des données et services disponibles. Il est alors possible de créer des applications pervasives de manière déclarative en utilisant des requêtes continues orientées services. Nous traitons des problématiques suivantes: 1) une représentation homogène de données et de services de l'environnement, 2) un langage de requêtes sur un environnement pervasif, 3) un système de gestion de l'environnement pervasif (PEMS). Cette démonstration présente notre prototype en action, permettant à l'utilisateur de visualiser l'environnement et de définir des applications pervasives.

MOTS CLES : Systèmes pervasifs, requêtes continues, services.

ABSTRACT
Pervasive systems give an overview of what digital environments should look like in the future. SoCQ takes a data-oriented perspective on the pervasive environment through a unified view of available data and services. Pervasive applications can then be created in a declarative fashion using service-oriented continuous queries. We tackle the following challenges: 1) a homogeneous representation for data sources and services from the pervasive environment, 2) a query language over pervasive environments, 3) a Pervasive Environment Management System (PEMS). This demonstration shows our PEMS prototype in action, allowing the user to visualize the environment and define pervasive applications.

CATEGORIES AND SUBJECT DESCRIPTORS: C.2.4 Distributed systems, H.2.4. Systems for database management.

GENERAL TERMS: Design, Experimentation.

KEYWORDS: Pervasive systems, continuous queries, services.

MOTIVATION
Our daily computing environment is composed of more and more heterogeneous computing devices [6], each one having a specific role. Alongside with personal computers and other handheld devices, sensors and actuators, be they physical or logical, are distributed in our environment and can provide to us useful data, transform raw data into more informative data, or perform some actions on the environment [1, 7]. Such an environment is full of functionalities, but a user may be lost and not able to comprehend and optimally use all available data sources and services the environment can provide. Furthermore, applications are not easy to develop and maintain because of the heterogeneity and the dynamicity of the environment. Typically, low-level technical code using programming languages (Java, C#...) and network protocols (JMX, UPnP...) has to be devised to come up with some pervasive applications.

The **S**ervice-**o**riented **C**ontinuous **Q**uery project, or SoCQ project [1], is devoted to making the development of pervasive applications easier through database principles. It aims at contributing in the area of Dataspaces [2, 5] through a unified view of data and service spaces mandatory in pervasive environments. We propose an approach [3, 4] to homogeneously represent such pervasive environments through database principles. The basic idea is to present to application developers a database-like view of the environment resources, so that they can visualize this environment as a set of tables and launch declaratively-defined continuous queries involving available data sources and services. This approach is built on an extension of the relational model and use a SQL-like query language.

We call PEMS, for Pervasive Environment Management System, a system that manages an environment containing data relations, data streams and services. Through this homogeneous relational representation, the different operations of the PEMS (resource discovery, service invoca-

[1] http://socq.liris.cnrs.fr

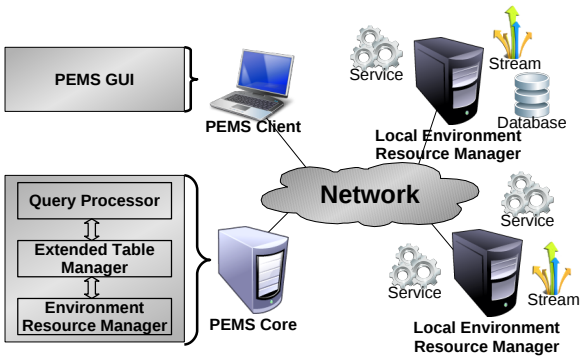

Figure 1 : Overview of a PEMS Environment

tions) can be translated into operations on tables (updating tables, retrieving values of virtual attributes, combining tables, etc.). Pervasive applications can then be expressed using declarative one-shot or continuous queries, and query optimization techniques can be applied to optimize the execution of those applications.

DEMONSTRATION OVERVIEW

In the demo scenario, we monitor temperatures in an office building: when a temperature exceeds some threshold in a room, an alert message is sent to the manager of this room. A photo of the room can be joined to the message. We simulate an environment containing the following data sources and services: 1) two data relations: information about the rooms (manager, temperature threshold...), list of contacts (including contacts for the managers), 2) temperature sensors distributed in several rooms, providing data streams, 3) cameras in the rooms, providing photo services, 4) messenger services (mail, instant message, SMS).

This environment can be represented homogeneously with relations and streams extended with virtual attributes and binding patterns [3, 4]. Virtual attributes are attributes that do not have a value and may be provided a value through a query, due to binding patterns that indicate their relationship with methods from services. We call such relations *XD-Relations*, standing for eXtended Dynamic Relations, and such environments *relational pervasive environments*. For the demo scenario, we can view a DDL representation of the schema of this environment in Table 1.

With such environments, the use of distributed functionalities provided by services is declaratively specified in SQL-like queries by the virtual attributes that need to be realized, *i.e.* that need to be provided a value. In order to realize those attributes, the corresponding binding patterns are invoked for every involved tuples, and results in different service invocations. We call these queries SoCQ, for Service-oriented Continuous Queries.

Over this environment, different SoCQ queries can be launched. The temperature monitoring can be declaratively defined as a continuous query. The three XD-Relations are joined (the stream "temperature" must be windowed) on the manager name and the area, the threshold is checked and the message body is set. The binding pattern "sendMessage" will be invoked in order to fetch a value for the initially virtual attribute "sent". The SQL-like query in Table 2 is a typical example of a pervasive application that is defined at the declarative level, without worrying about low-level technical considerations (programming languages, network protocols).

Table 1 : DDL description of XD-Relations for the environment of the demo scenario

```
RELATION surveillance (
  area         STRING,
  manager      STRING,
  threshold    REAL,
  alertMessage STRING
);

RELATION employees (
  name      STRING,
  address   STRING,
  messenger SERVICE,
  text      STRING   VIRTUAL,
  sent      BOOLEAN  VIRTUAL
)
USING BINDING PATTERNS (
  sendMessage[messenger] ( address, text ) : ( sent )
);

RELATION cameras (
  camera  SERVICE,
  area    STRING,
  photo   BINARY  VIRTUAL
)
USING BINDING PATTERNS (
  takePhoto[camera] ( ) : ( photo )
);

STREAM temperatures (
   area        STRING,
   temperature REAL
);
```

Table 2 : A query of the demo scenario

```
SELECT surveillance.area, surveillance.manager, employees.sent
FROM   temperatures [now], employees, surveillance
WHERE  surveillance.manager = employees.name
  AND  surveillance.area = temperatures.area
  AND  surveillance.threshold < temperatures.temperature
  AND  employees.text IS surveillance.alertMessage
USING sendMessage
```

The role of a PEMS is to manage a relational pervasive environment, with its dynamic data sources and set of services, along with the execution of the continuous queries over this environment. In the following sections, we first sketch the data model that supports XD-Relations, and the algebra that enables SoCQ queries. We then detail the demonstration scenario.

MODELING OF PERVASIVE ENVIRONMENTS

Our model, based on the relational model, is built on the following notions: prototypes, services and extended relations with virtual attributes and binding patterns [3, 4]. Distributed functionalities can be represented as services implementing prototypes. For example, a webcam and an IP camera are two services from the environment that implement a prototype `takePhoto():(photo)` that takes zero input attribute and provides one output attribute `photo`; a mail server, an instant messaging server and a SMS gateway are three services that implement a proto-

type `sendMessage(text,address):(sent)` that takes two input attributes `text` and `address` and provides one output attribute `sent`. Invoking a prototype on a service realizes the implied actions, like taking a photo for a camera and sending a message to the given address for the mail server.

Prototypes can be integrated into data relations schemas through virtual attributes and binding patterns. Virtual attributes are attributes from the relation schema that do not have a value at the tuple level. They represent input and output attributes of prototypes. A binding pattern is associated with a relation schema and specifies one non-virtual attribute from the relation schema that is the service reference, the prototype and which attributes are linked with the prototype input and output attributes. For example, the `employees` relation in Table 1 is associated with one binding pattern that uses the prototype `sendMessage`, the service reference attribute `messenger` and that links the attributes `address` and `text` with the prototype input attributes, and the attribute `sent` with the prototype output attribute. Output attribute should be virtual attributes, whereas input attributes can also be real (*i.e.* , non-virtual) attributes, like the attribute `address` in this example.

We call such relations, X-Relations, standing for eXtended Relations. Virtual attributes represent possible interactions with services: when a query needs the virtual attribute `sent`, a value is required for the virtual attribute `text` due to the binding pattern (the attribute `address` being real), and it implies an invocation of the prototype `sendMessage`. The required value should be provided by the query itself. The services on which the prototype is invoked are defined by the value of the service reference attribute (here, attribute `messenger`), at the tuple level.

In the following table, an example of content for the X-Relation `employees` is presented. The constants "mailer" and "jabber" are two service references, the former for the mail server, the latter for the instant messaging server. The star (*) symbol reminds that virtual attributes have no value.

name	address	messenger	text	sent
nicolas	nicolas@elysee.fr	mailer	*	*
carla	carla@elysee.fr	mailer	*	*
francois	francois@im.gouv.fr	jabber	*	*

Pervasive environments being dynamic, data sources may include streaming data. We extend our model to integrate data sources like data streams. We call XD-Relations, for eXtended Dynamic Relations, X-Relations that are time-dependent: XD-Relations can be either finite (relations where tuples can be inserted and deleted) or infinite (append-only relations, *i.e.* data streams). An environment represented by a set of XD-Relations is defined as a relational pervasive environment.

SERVICE-ORIENTED CONTINUOUS QUERIES

Queries over relational pervasive environments allow to define interactions between dynamic data sources and services, *i.e.* pervasive applications. Such queries are defined to be continuous queries, *i.e.* queries that are executed continuously to maintain their results up-to-date, like in the demo scenario. They are called Service-oriented Continuous Queries, or SoCQ queries. However, some queries may be snapshot queries, *i.e.* queries executed once that produce their results and do not maintain them, like standard SQL queries in DBMS.

SoCQ queries are based on the Serena algebra [4] (**Ser**vice-**ena**bled relational algebra) that defines query operators over XD-Relations. Standard relational operators are redefined over finite XD-Relations, and new operators are defined. Realization operators handles the transformation of virtual attributes either by providing them a value (a constant or the value of another attribute) or by invoking a binding pattern. Window operators and streaming operators handles infinite XD-Relations: window operators transform an infinite XD-Relations into a finite XD-Relations (*e.g.* a relation that contains the tuples inserted during the last 5 minutes into the stream operand), and streaming operators transform finite XD-Relations into infinite XD-Relations (*e.g.* a stream of the tuple inserted into the relation operand).

A SQL-like query language has been defined to declaratively express SoCQ query. For example, for the demo scenario, the query in Table 2 involves several operators: windows (the `[now]` is a window of size 1 applied on the stream `temperatures`), selections, joins, realizations, streaming. This query produces a stream of alerts (when a threshold is exceeded) while invoking the `sendMessage` prototype when needed (to send message to area managers).

IMPLEMENTATION OF PEMS

The PEMS core (see Figure 1) is composed of three modules. A global resource manager handles service discovery and remote invocations, with local resource managers as distributed proxies for local devices that provide services. An extended table manager builds a homogeneous representation of non-conventional data sources, and the query processor allows to define, optimize and execute Service-oriented Continuous Queries.

The PEMS prototype is developed in the Java/OSGi framework [2]. Each module of the PEMS is an OSGi bundle and communicates with each other through the OSGi service lifecycle management. The chosen network protocol for service discovery and remote invocations is UPnP [3]: the prototype uses the dedicated standard OSGi bundles for this protocol.

[2] `http://www.osgi.org`
[3] `http://www.upnp.org`

The PEMS GUI is also developed in the Java/OSGi framework, as an Eclipse RCP Plugin, *i.e.* the GUI is integrated in the Eclipse platform. It communicates remotely with the PEMS core through a JMX interface. It enables to visualize existing XD-Relations and their content, to add/alter/delete XD-Relations, and to launch/stop SoCQ queries.

DEMONSTRATION SCENARIO

The demonstration platform is a functional PEMS and a "pervasive" environment that provides services and data streams mentioned in this paper. A first PC is used to host the PEMS and to interact with it through its GUI. Two other PCs are used to host different services and devices: physical or simulated temperature sensors (*Thermochron® iButton® DS1921*), webcams (from *Logitech®*), instant messaging server (*Openfire* server from *Jive Software®*), (gateway to) SMS gateway (commercial service from *Clickatell*), (gateway to) mail server. Two handheld devices (1 PDA, 1 SmartPhone) are also used to host instant messaging client.

A PEMS environment configuration, *i.e.* data sets and SoCQ queries, is provided to demonstrate the scenario mentioned in the paper. This environment configuration can be modified, resetted and reloaded when needed.

A demo visitor interacts with the platform in three steps:

1. *Testing the demo configuration:* visualize the environment through the PEMS GUI and observe the execution of the demo query while manipulating the sensors to interact with the environment.
2. *Modifying the demo configuration:* modify the data and/or the query and observe the impact on the pervasive application.
3. *Creating a new configuration:* create data and queries from an empty configuration to create some pervasive applications.

The goal of the first step is to present the model for relational pervasive environment and SoCQ queries. We visualize the environment like a database: XD-Relations are represented like tables, that contain tuples. Alongside the UML-like graphical representation of their schema, the source DDL can be viewed. SoCQ queries are represented in a similar way to tables: the schema of their resulting relation is shown, and the query source can also be viewed. The demo visitor can manipulate the physical sensors (*e.g.* heating the temperature sensors in his hand) and the simulated sensors (through another GUI) in order to test the behavior of the pervasive application defined by the SoCQ query.

The second step consists in modifying the data and/or the query. By modifying the data, the behavior of the pervasive application can be slightly modified, *e.g.* by lowering a temperature threshold or changing the alert message. The demo visitor is invited to enter his own contact information to be able to receive a SMS sent by the query (through the provided SMS gateway service). By modifying the query, the behavior of the pervasive application can be totally changed. The demo visitor also has the opportunity to pose his own SoCQ queries.

The final step gives the demo visitor a complete freedom to create a relational pervasive environment from scratch, *i.e.* to create XD-Relations, populate them with data, and test SoCQ queries. He is invited to use all the available services and data sources to build his own environment and pervasive applications (*e.g.* the camera services are available but not involved in the demo configuration).

DEMONSTRATION CONTRIBUTIONS

This demo allows to demonstrate the following points: 1) a homogeneous database-like view on pervasive environments containing dynamic data sources and services is possible as a set of XD-Relations, through the notions of virtual attributes and binding patterns; 2) Service-oriented Continuous Queries (SoCQ queries) over relational pervasive environment allow to define pervasive applications combining data sources and services; and 3) declarative definitions of SoCQ queries make the definition and the evolution of pervasive applications easier. Although this model is presented in the context of pervasive environments, it can be applied to less dynamic systems that could benefit from the declarative definition of applications that combine data sources and distributed functionalities.

BIBLIOGRAPHIE

1. Estrin, D., Culler, D., Pister, K., and Sukhatme, G. Connecting the Physical World with Pervasive Networks. *IEEE Pervasive Computing*, 1(1):59–69, 2002.
2. Franklin, M., Halevy, A., and Maier, D. From Databases to Dataspaces: a new Abstraction for Information Management. *SIGMOD Rec.*, 34(4):27–33, 2005.
3. Gripay, Y. Service-oriented Continuous Queries for Pervasive Systems. In *EDBT 2008 PhD Workshop*, 2008.
4. Gripay, Y., Laforest, F., and Petit, J.-M. Vers une algèbre relationnelle étendue aux services. In *BDA 2008*, pages 1–20, Oct. 2008.
5. Imielinski, T., and Nath, B. Wireless graffiti: data, data everywhere. In *VLDB'2002*, pages 9–19, 2002.
6. Weiser, M. The Computer for the 21st Century. *Scientific American*, 265(3):94–104, September 1991.
7. Zhu, F., Mutka, M., and Ni, L. Service Discovery in Pervasive Computing Environments. *IEEE Pervasive Computing*, 4(4):81–90, 2005.